Robotic Uniform Coverage
of Arbitrary–Shaped Connected Regions

Paul M. Maxim and William M. Spears

Abstract—In this article we present a novel algorithm for uniform coverage of a region. Surveillance, cleaning, and mine detection are some applications that would benefit from this type of algorithm. Prior work has claimed that this task is impossible to solve for non–convex regions [7]. Our algorithm enables robots to uniformly cover arbitrary path–connected regions, such that the robot movements are not predictable and the region periphery is not neglected. The algorithm assumes that robots are independent and is physics–based, relying on an analogy with mean free paths of particles. Validation of the algorithm is rigorously provided via simulation and real robot experiments.

Index Terms—uniform coverage, arbitrary–shaped connected regions, mean free path, Markov chains, robot experiments.

I. INTRODUCTION

Robotic coverage can be defined as the problem of moving a sensor or actuator over all points in a given region [10]. Robotic de–mining operations, snow removal, lawn mowing, car–body painting, reconnaissance, surveillance, search and rescue operations, and ship hull cleaning are some of the multitude of tasks that coverage algorithms can be applied to. A coverage algorithm must generate what is called a coverage *path*, which is a sequence of motion commands for a robot. These algorithms can be classified as either complete or heuristic and randomized.

By definition, complete algorithms guarantee a path that completely covers a path–connected [16], robot–traversable region [4]. Heuristic algorithms on the other hand, use simple rules that may work very well, but they have no provable guarantees to ensure the success of coverage.

Some of the existing complete algorithms decompose the region to be covered into subregions that are simple to cover, then a travelling salesman algorithm is applied to generate a coverage path that covers each subregion in turn [10], [4]. Another approach is to create an artificial potential field, where a numerical value is assigned to each cell. This value is then increased every time there is a visit to the cell. The next step is then taken to the cell with the lowest value among the adjacent cells. Real–time complete coverage can also be achieved by making the robot mark its path, and then avoid areas previously marked [6]. The algorithms require prior knowledge of the shape of the environment and/or the ability to change the environment, either physically or virtually.

Heuristic algorithms for coverage employ simple behaviors, such as spiraling, room crossing, or wall following [11].

P. M. Maxim is with ITS/GIS Program, Wyoming Department of Transportation, Cheyenne, WY 82009, USA, e-mail: paul.maxim@dot.state.wy.us

W. M. Spears is with Swarmotics LLC, Laramie, WY 82070, USA, e-mail: wspears@swarmotics.com

More complicated actions, like explorations, can be achieved with a hierarchy of cooperating behaviors. Although complete coverage is not guaranteed, there are advantages to this class of algorithms. A cost/benefit analysis for randomized search has been performed separately, by Balch [2] and Gage [7]. Robots using randomized search strategies can be less expensive than robots using methods that need precise localization hardware. Balch found that it may be effective to use a randomized search if the robots without localization capabilities can be constructed at one fifth the cost of the more expensive robots.

Gage focuses on coverage for de–mining purposes and he takes into consideration the probability of a robot's sensor to detect a mine. The probability of correctly detecting a mine is directly proportional to the advantages of methodical search algorithms over the randomized search algorithms. When these advantages eventually disappear, the use of less expensive hardware is preferable.

Gage introduces a randomized search strategy that provides uniform coverage [7] and defines certain properties that a path generator algorithm must take in consideration: (1) the path should not be predictable (so that an observer cannot predict the robot's future path by knowing its prior path), (2) the path must not be determined by a fixed reaction to environmental features, and (3) the periphery of the search space should not be neglected. The algorithm presented is based on diffuse reflection – how light reflects off a matte surface – which is then generalized to any convex search areas. However, in order to implement this algorithm, "the searcher *must* be able to determine its position within the search area, and the distance to the opposite boundary in all directions". Gage states that it is impossible to extend this strategy to generalized non–convex areas, unless more restrictions are added.

We have invented a novel, yet simple, algorithm that meets Gage's three properties, but works in a broad class of environments that can be tiled with square cells. The environment must be path–connected and robot–traversable, but can be convex or concave. The algorithm is randomized and complete.

Section II introduces the environment generation tool, the simulation tool we developed to test our algorithm, the performance metric, and the test environments. Two versions of our uniform coverage algorithm are also presented in this section. In Section III we present the theory that supports our algorithm. Validation of our algorithm with actual robot experiments is presented in Section VI. Section VII summarizes this article.

Report Documentation Page

Public reporting burden for the collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to a penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.

1. REPORT DATE **JAN 2010**	2. REPORT TYPE	3. DATES COVERED **00-00-2010 to 00-00-2010**

4. TITLE AND SUBTITLE **Robotic Uniform Coverage of Arbitrary-Shaped Connected Regions**	5a. CONTRACT NUMBER
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Swarmotics LLC,Laramie,WY,82070**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT
In this article we present a novel algorithm for uniform coverage of a region. Surveillance, cleaning, and mine detection are some applications that would benefit from this type of algorithm. Prior work has claimed that this task is impossible to solve for non?convex regions [7]. Our algorithm enables robots to uniformly cover arbitrary path?connected regions, such that the robot movements are not predictable and the region periphery is not neglected. The algorithm assumes that robots are independent and is physics?based, relying on an analogy with mean free paths of particles. Validation of the algorithm is rigorously provided via simulation and real robot experiments.

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**	**Same as Report (SAR)**	**15**	

II. SIMULATION

A. Environment Generator

Our environment generator allows us to control the shape and the size of the environment. To be able to monitor how uniformly the robot is covering the environment, we split the environment into equal–size square cells (Figure 1). For example, a square environment that is 60×60 units can be split into 9 equal cells, each cell being 20×20 units (we will henceforth adopt the convention that a cell that is $c \times c$ units will be referred to as a cell of size c). This way we can monitor how often each cell was visited and measure how well the algorithm is performing. To ensure that our algorithm is scale invariant, we also test on scaled–down versions of the same environment, as we explain next.

0	1	2
3	4	5
6	7	8

Fig. 1.   A 60×60 units environment tiled with nine cells of size 20.

We created a graphical environment generator tool that allows a user to specify the initial cell size and its decrement factor. The decrement factor is used to generate scaled–down versions of the generated environment. For example, if a cell size of 30 units is chosen with a decrement factor of 5 units, a total of 5 environments will be generated, having cell sizes 30, 25, 20, 15, and 10 units. Note, this does not change the number of cells, since the whole environment is scaled the same way. The Graphical User Interface (GUI) presents the user a square window with a grid whose vertices are equally placed at the initial cell size intervals. The user then clicks on the vertices of the desired environment. When all the vertices are selected, a right click anywhere on the grid will cause the tool to generate all the environments and then exit. A GUI snapshot showing an arbitrary environment outline is presented in Figure 2 (the generated output is presented in Listing 7 from Appendix A).

The output files are named *environment_X.ini*, where *X* represents the cell size. Using the same example as above, the output files are: *environment_30.ini*, *environment_25.ini*, *environment_20.ini*, *environment_15.ini*, and *environment_10.ini*. Each of these files contains the total number of vertices the environment has, the coordinates of each of these vertices, the total number of cells and the lower–left corner coordinates of each of the cells, as well as the length of one cell side. The 60×60 environment with nine cells of size 20 is presented in Listing 1.

B. Simulation Tool

Our simulation is programmed in the C language and we use the Free OpenGL Utility Toolkit (Freeglut) version 2.4.0 library [19] to add graphic display capabilities. Freeglut is an open–source alternative to the OpenGL Utility Toolkit (GLUT) library. We run our simulation on platforms running Linux–based operating systems, but it can easily be ported to different platforms and operating systems.
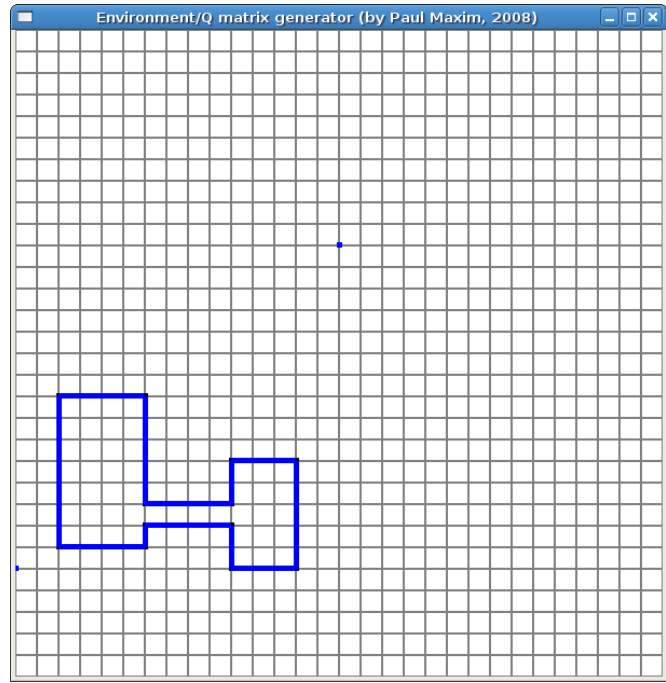

Fig. 2.   Environment generator snapshot

Listing 1.   *environment_20.ini* example

```
      vertex_number 4
      0         0
      0         60
      60        60
5     60        0
      cell_number    9
      0         0
      0         20
      0         40
10    20        0
      20        20
      20        40
      40        0
      40        20
15    40        40
      cell_size      20
```

1) Configuration and control: The configuration and control of the simulation tool can be performed at two levels: before and during the simulation. Before the simulation is started, a configuration file allows the user to set certain parameters to their desired values. Some of these parameters control the simulation environment, like the size of the graphics window, the simulation speed, and the simulation length, while other parameters control the actual algorithm and the robot, or agent, being simulated. Examples are the obstacle detection sensor range and the distance at which the robot will react to the obstacles.

The actual simulation window has three distinct areas. The top area displays important real–time data about the current simulation, the middle area is the largest and it shows the outline of the environment together with the robot that navigates through the environment. The robot is shown as a blue disk, with a black dot marking its current heading and a red line that shows the position and the current range of the
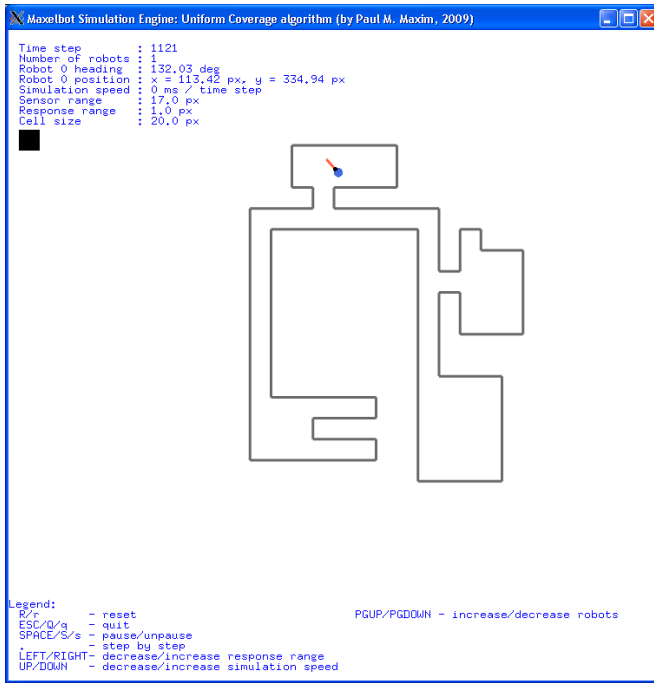
Fig. 3.   Uniform coverage simulation engine snapshot featuring a complex 84 cell environment (cell size of 20)

obstacle detection sensor. A legend at the bottom of the screen informs the user about the commands that can be performed while the simulation is running, like increasing or decreasing the simulation speed or the obstacle detection sensor response range. Figure 3 shows a snapshot of the simulation window that features a complex 84 cell environment. In this figure, the robot is shown inside the top chamber of the environment.

A file with the environment configuration must be provided to the simulation tool. For our preliminary experiments a robot is run for 10 million time steps inside this environment, recording to a file (every 10,000 time steps) the number of visits the robot has made to each of the cells in the environment (the final distribution of visits per cell is what we are most interested in). Initially, for each environment we varied the starting position of the robot from being in a corner, a side, or an interior cell. Also the obstacle detection sensor response range was set to 1.0 unit to meet Gage's condition that the periphery should not be neglected. To avoid wall collisions the robot velocity was set to 0.8 units/step.

C. Kullback–Leibler Divergence

After we run our simulation tool on an environment we get a distribution that shows us how many visits each cell in the environment has. Rather than just rely on visual inspection, a metric is necessary to formally measure the difference between this observed distribution and the optimum uniform distribution.

In probability theory, the Kullback–Leibler divergence is a non–commutative measure of the difference between two probability distributions P and Q. Typically P represents the "true" distribution of data, observations, or a precise calculated

theoretical distribution. The measure Q typically represents a theory, model, description, or approximation of P [12].

In our case P represents the observed distribution and Q represents the optimum uniform distribution. The minimum divergence is 0.0, meaning the observed distribution is precisely the same as the uniform distribution. The K–L divergence of Q from P is defined to be:

$$D_{KL}(P\|Q) = \sum_i P(i) log \frac{P(i)}{Q(i)} \qquad (1)$$

In general, a K–L divergence of 0.01 is considered to be an excellent match between the two distributions [1]. For our results with real robots our goal is for a K–L divergence of approximately 0.01. For simulation results our goal is 0.001 since we can run the simulation for a far greater number of steps.

D. Markov Chain Analysis

Markov chains are used in mathematics to describe the future states of a stochastic process based on the present state [8]. The future states are independent of the past state and are reached through a probabilistic process. A change of a state is called a transition and the probability that governs this change is called a transition probability. A probability matrix Q defines the probability of transitioning to a state j in the next step, given the system is currently in state i (i.e., $Q(i,j)$).

A Markov chain is ergodic if all states are recurrent, aperiodic, and communicate with each other. A state i is transient if there exists a state j that is reachable from i, but state i is not reachable from j. If a state is not transient, it is recurrent. Also, a state i is periodic if all paths leading from i back to i have a length that is multiple of some integer $k > 1$. If this is not the situation, the state is aperiodic. All states communicate with each other if there is a path from any state i to any state j (and vice versa).

Every ergodic Markov chain has a steady–state distribution that describes the long–term behavior of the system. The existence of a steady–state distribution indicates that the initial state of the system is of no importance in the long term. Furthermore, if Q is symmetric ($Q(i,j) = Q(j,i)$), then the steady–state distribution is the uniform distribution. This latter property is crucial to achieving a good uniform coverage algorithm.

Here we assume that a state is simply the cell that the robot is in. The robot transitions from cell to cell as it moves. Hence, if our algorithm for moving the robot can be described as an ergodic Markov chain with a symmetric probability transition matrix, the steady–state distribution must be uniform and the algorithm is correct. The algorithm is also complete if the environment is path–connected and robot–traversable.

Consider the nine–cell environment shown in Figure 1. Let us also consider an idealized situation where a robot can reside at the center of cells. For example, suppose the robot is at the center of cell 4. The robot could then transition to adjacent cells 1, 3, 5, or 7. The probability of transitioning directly to cells 0, 2, 6, or 8 is essentially zero (i.e., the robot would have to pass through cells 1, 3, 5, or 7 first to get to the

other cells). Hence we will ignore the "diagonal" neighbors. An ideal algorithm is as follows. The robot turns uniformly randomly and then attempts to move. It might stay in the same cell (if the cell is along the border of the environment), or it might enter an adjacent cell. If it enters an adjacent cell we assume it then moves to the center of that cell, so that the algorithm can continue.

The Markov chain for this idealized algorithm is trivial to compute. If a cell has b periphery edges, then there is a $b/4$ probability of staying in that cell. The remaining probability mass is uniformly distributed to adjacent cells.

The probability transition matrix Q for the nine–cell environment is:

$$\begin{bmatrix} 0.50 & 0.25 & 0.00 & 0.25 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\ 0.25 & 0.25 & 0.25 & 0.00 & 0.25 & 0.00 & 0.00 & 0.00 & 0.00 \\ 0.00 & 0.25 & 0.50 & 0.00 & 0.00 & 0.25 & 0.00 & 0.00 & 0.00 \\ 0.25 & 0.00 & 0.00 & 0.25 & 0.25 & 0.00 & 0.25 & 0.00 & 0.00 \\ 0.00 & 0.25 & 0.00 & 0.25 & 0.00 & 0.25 & 0.00 & 0.25 & 0.00 \\ 0.00 & 0.00 & 0.25 & 0.00 & 0.25 & 0.25 & 0.00 & 0.00 & 0.25 \\ 0.00 & 0.00 & 0.00 & 0.25 & 0.00 & 0.00 & 0.50 & 0.25 & 0.00 \\ 0.00 & 0.00 & 0.00 & 0.00 & 0.25 & 0.00 & 0.25 & 0.25 & 0.25 \\ 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.25 & 0.00 & 0.25 & 0.50 \end{bmatrix} \quad (2)$$

This matrix is ergodic and the steady–state distribution indicates that each cell is visited precisely $1/9^{th}$ of the time, which is what we desire.

It is important to point out that this is an existence proof that an algorithm may exist that provides uniform coverage for all path–connected environments that can be tiled with square cells. The algorithm given above is not realistic. However, the argument above does provide guidance on how to create a viable algorithm: the resulting Markov chain must be ergodic and have a symmetric probability transition matrix.

E. First Algorithm

Our first algorithm requires only that the robot be able to sense an object (wall) in front of it. The robot starts by moving forward. When it senses an object directly in front of it, the robot stops. Then the robot makes a uniform $[-\pi, \pi]$ random turn. If there is still an object in front, the robot turns again. The robot continues to turn until it detects no obstacle in front of it. Then the robot moves forward again. This algorithm does not avoid the periphery, it is not predictable, and it works in all path–connected environments. Proof of this claim lies in the assumption of tiling the environment with square cells. The resulting Markov chain is ergodic and will have a steady–state distribution. However, it is not clear whether the steady–state distribution is uniform.

F. Environments

The first test of our algorithm was on a simple 40×40 units square environment, with four cells of size 20. As seen in Listing 2, the distribution is very close to uniform. This is reasonable, since all four cells are corner cells and can be considered equivalent.

Listing 2.   Visits in the four cell square environment

```
cell 0: 2503625
cell 1: 2500184
cell 2: 2494401
cell 3: 2501790
```

The next tested environment is the 60×60 units square environment shown in Figure 1, with nine cells. This provides us with different categories of cells, such as corners, sides, and an inner cell. The results for this environment are presented in Listing 3. We can see that the distribution is *not* uniform. In fact, the center cell (4) has the least amount of visits. The main reason is the inner position of this cell. The robot does not detect any obstacles in this cell and spends the least amount of time there because it moves straight through it. Clearly, this first algorithm is not adequate, since the probability of transitioning from the center cell to any other cell is higher than transitioning from any other cell to the center cell. A similar difference exists between the corner cells and side cells.

Listing 3.   Visits in the nine cell square environment (1st algorithm)

```
cell 0: 1242516
cell 1: 1048036
cell 2: 1229541
cell 3: 1042531
cell 4:  888209
cell 5: 1052790
cell 6: 1225855
cell 7: 1041632
cell 8: 1228890
```

Taking random turns only when detecting a wall is not sufficient. This tells us that in certain situations, $Q(i,j) \neq Q(j,i)$. An improvement has to be brought to this algorithm. This improvement is inspired from physics, as we describe next.

G. Second Algorithm

The prior results on the nine cell environment indicate that the transition probabilities are not symmetric. In fact, the results indicate that the robot needs to spend more time in cells with less peripheral edges. This can be achieved using the concept of *mean free path* from physics. In physics, a molecule in a gas moves with constant speed along a straight line between successive collisions. The mean free path is defined as the average distance between such successive collisions [9]. Depending on the number of molecules in the space available to them, the mean free path can range from zero to infinite.

In the first algorithm, the only time the robot changes direction is when it senses a wall. Hence it spends less time in cells with fewer peripheral edges. In order to counteract this, we want the robot to act as if it collides with another virtual object occasionally. Hence we want the robot to randomly change direction after it has moved a certain distance, namely, its mean free path. Since the mean free path needs to be normalized by the cell size, we assume the mean free path

will be $f \times c$, where f is some real number, and c is the cell size.

We empirically determined that $f \approx 0.6$ works best. If we rerun the same experiment with the nine–cell environment, the results are shown in Listing 4. We can easily see the improvement of this version of the algorithm over the previous version. By using the Kullback–Leibler (K–L) divergence metric we can measure how close this distribution is to the uniform distribution. For this experiment the K–L divergence is 0.000060. Remember that the closer this metric is to 0.0, the better. To give a better understanding of how the K–L metric changes over time, Figure 4 shows the graph of the K–L divergence metric every 10,000 time steps for 10 million time steps. This graph is not smooth because it represents only one simulation run.

Listing 4.   Visits in the nine–cell square environment (2nd algorithm)

```
cell 0: 1098245
cell 1: 1110451
cell 2: 1112045
cell 3: 1124196
cell 4: 1127813
cell 5: 1124090
cell 6: 1089300
cell 7: 1111144
cell 8: 1102716
```
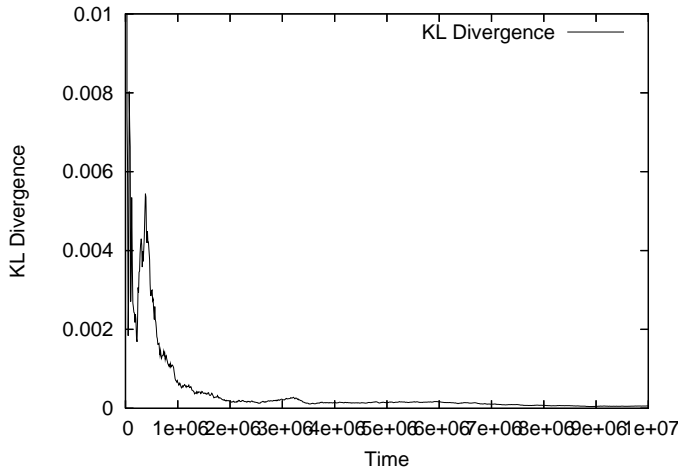

Fig. 4.   K–L divergence metric for the nine–cell square environment

It is important to note that because the Markov chain is ergodic the time average of our algorithm converges to the ensemble average that would be computed by running the algorithm multiple times [21]. Hence, it isn't necessary to perform multiple runs – the long term behavior of the algorithm represents the ensemble average and more realistically describes the behavior of a robot, which we assume has a very long–term presence in the environment, performing the desired coverage task.

III. DERIVATION OF THE MEAN FREE PATH

The mean free path can be thought of as the average distance travelled by a robot to move from one cell to another adjacent cell. Consider Figure 5, with two adjacent cells. Each cell is 1×1 units in size. Without loss of generality, assume robot A is located inside the left cell at coordinates (x_1, y_1). We would like to compute the average distance travelled by the robot to barely enter the right cell at location $(1 + \epsilon, y_2)$.

Fig. 5.   What is the average distance travelled by A to enter the right cell?

The simplest way to compute this distance is via simulation as shown in Listing 5. Function *u01()* returns floating point values uniformly randomly from 0 to 1. This simulation yields a mean free path of 0.6517.

Listing 5.   Mean free path simulation – C code

```c
int main( int argc, char** argv ) {
  int i, N;
  double sum, x1, x2, y1, y2;
  N = 10000000;
  sum = 0.0;
  for (i = 1; i <= N; i++) {
    x1 = u01();
    x2 = 1.001;
    y1 = u01();
    y2 = u01();
    sum = sum + sqrt((x2 - x1)*(x2 - x1) + \
                     (y2 - y1)*(y2 - y1));
  }
  printf("Average length = %f\n", sum/N);

  return 0;
}
```

A more formal derivation can be accomplished by computing the average distance using a definite integral:

$$\int_0^1 \int_0^1 \int_0^1 \sqrt{((1 - x_1)^2 + (y_1 - y_2)^2)} \, dx_1 dy_1 dy_2 \quad (3)$$

The solution to the indefinite integral [18] is:

$$
\begin{aligned}
F(x_1, y_1, y_2) = \Big[& 3y_2{}^4 - 28y_1 y_2{}^3 + 42y_1{}^2 y_2{}^2 - \\
& 36(1 - x_1)y_2{}^2 \sqrt{y_2{}^2 - 2y_1 y_2 + (1 - x_1)^2 + y_1{}^2} - 28y_1{}^3 y_2 + \\
& 48(1 - x_1)^3 y_2 \ln\left(-y_2 + y_1 + \sqrt{y_2{}^2 - 2y_1 y_2 + (1 - x_1)^2 + y_1{}^2}\right) + \\
& 72(1 - x_1)y_1 y_2 \sqrt{y_2{}^2 - 2y_1 y_2 + (1 - x_1)^2 + y_1{}^2} - \\
& 12(y_2 - y_1)^4 \ln\left(1 - x_1 + \sqrt{y_2{}^2 - 2y_1 y_2 + (1 - x_1)^2 + y_1{}^2}\right) + \\
& 48(1 - x_1)^3 y_1 \ln\left(y_2 - y_1 + \sqrt{y_2{}^2 - 2y_1 y_2 + (1 - x_1)^2 + y_1{}^2}\right) + \\
& 24(1 - x_1)^3 \sqrt{y_2{}^2 - 2y_1 y_2 + (1 - x_1)^2 + y_1{}^2} - \\
& 36(1 - x_1)y_1{}^2 \sqrt{y_2{}^2 - 2y_1 y_2 + (1 - x_1)^2 + y_1{}^2} \Big] \left[\frac{-1}{288}\right]
\end{aligned}
$$

$$(4)$$

Given this, we can evaluate the definite integral to be [17]:

$$F(1,1,1) - F(1,0,1) - F(0,1,1) + F(0,0,1) -$$
$$F(1,1,0) + F(1,0,0) + F(0,1,0) - F(0,0,0) = .651757 \quad (5)$$

This conforms with the value computed using Listing 5. In experiments, we noted that $f = 0.60$ works very well, which is slightly lower than theory. The difference is caused by the fact that the robot must stay 1 unit away from the walls, which in turn makes the length of the mean free path slightly smaller. For example, we generally use cells of size 20. Thus, having a wall response range of 1 unit provides a 5% buffer zone. Assuming a 5% buffer, the theoretical mean free path is lowered to 0.6079, which is extremely close to our empirically derived value.

It is important to note that the concept of mean free passage has allowed us to make the probability transitions symmetric ($Q(i,j) = Q(j,i)$). This can be observed by noting that robot A could be in any cell of any square–tiled environment, and the mean free passage is the average distance travelled to enter any adjacent cell, regardless of whether these cells are on the periphery or not.

IV. FURTHER CONFIRMATION

We ran our simulation over a large variety of environments and we generally obtain a K–L divergence number of 0.001 or lower (the mean over all environments is 0.00066 with a standard deviation of 0.00068), which indicates that the behavior is extremely close to the optimum. Each simulation is run 40 million time steps. Table I provides results over numerous environments and cell sizes. The environment outlines are presented in Figures 21, 22, and 23 from Appendix A.

TABLE I
K–L DIVERGENCE METRIC FOR DIFFERENT ENVIRONMENTS AND CELL SIZES

Environment	Cell Size (units)				
	10	15	20	25	30
3x1	0.00011	0.00005	0.00003	0.00003	0.00002
4cellL	0.00026	0.00022	0.00019	0.00016	0.00022
4cellRoll	0.00090	0.00082	0.00068	0.00066	0.00071
7cell	0.00078	0.00075	0.00064	0.00056	0.00061
3x3	0.00018	0.00010	0.00007	0.00007	0.00006
15cell	0.00019	0.00010	0.00015	0.00016	0.00019
22cell	0.00088	0.00124	0.00188	0.00137	0.00145
35cell	0.00138	0.00079	0.00067	0.00067	0.00106
6x6	0.00021	0.00011	0.00009	0.00008	0.00009
82cell	0.00223	0.00114	0.00269	0.00097	0.00059
84cell	0.00210	0.00309	0.00067	0.00156	0.00064
10x10	0.00051	0.00057	0.00009	0.00057	0.00065

Figure 6 shows snapshots in time with one simulated robot uniformly covering the rather complex 84 cell environment. The grey–scale denotes the frequency that a particular cell is occupied (darker means less often). The uniformity of the grey cells in the last snapshot (lower right) indicates excellent uniform coverage. The K–L divergence graph for this environment is shown in Figure 7 and at the last step in the simulation its value is 0.00067.

(a) – 10,000 time steps

(b) – 100,000 time steps

(c) – 1,500,000 time steps

(d) – 5,000,000 time steps

(e) – 10,000,000 time steps
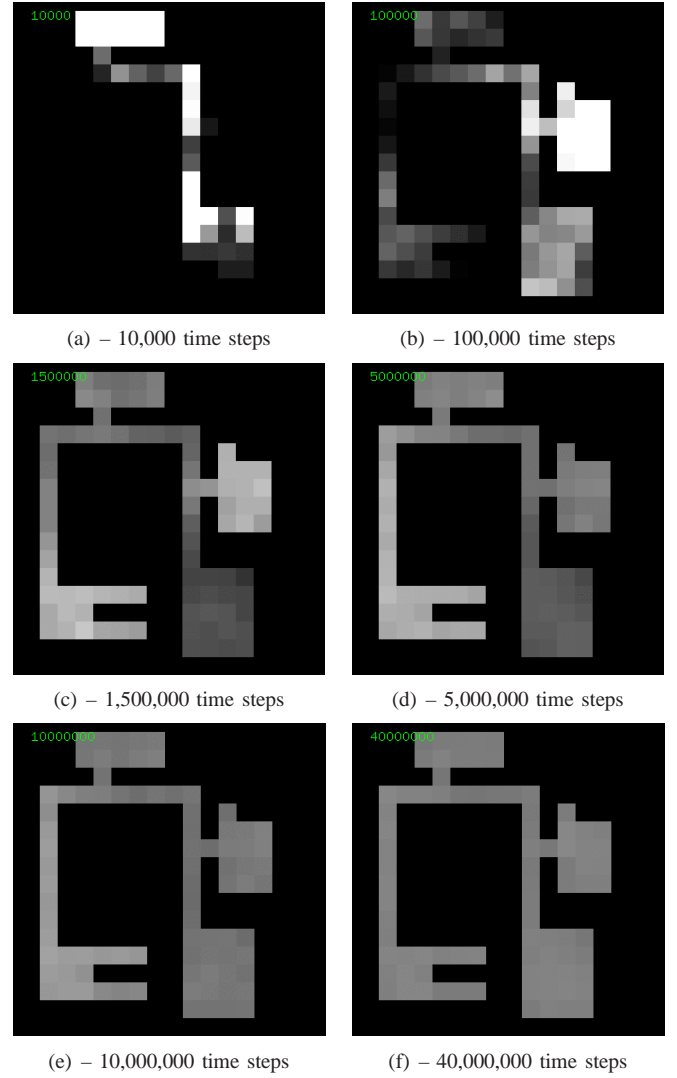
(f) – 40,000,000 time steps

Fig. 6. One simulated *Maxelbot* obtaining uniform coverage in a complex 84 cell environment – snapshots taken at the time steps shown above
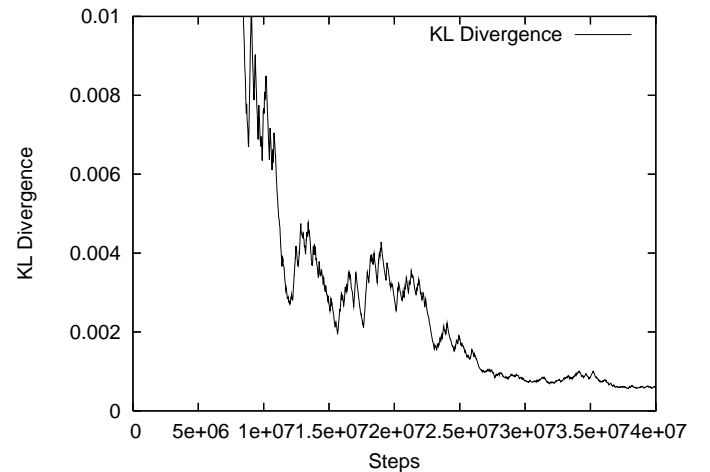

Fig. 7. K–L divergence metric for a simulated 84 cell environment

V. THEORY FOR MULTIPLE ROBOTS

One robot does very well at uniformly covering an environment using our algorithm. We have shown that, if a connected environment is composed of C square cells, in the long term there is a $\frac{1}{C}$ probability of the robot being in each cell.

Let us assume we have a team of N independent robots performing the same algorithm in a large environment, such that their interactions are minimal (they merely have to avoid each other on rare occasions). For example, $N \in [10, 50]$ is very reasonable for systems that are usually deployed in the field. The probability of one robot being in one of the cells is $\frac{1}{C}$. The probability of one robot *not* being in that cell is $1 - \left(\frac{1}{C}\right)$. The probability of all N robots *not* being in that cell is then $\left(1 - \left(\frac{1}{C}\right)\right)^N$. Hence, the probability that at least one robot is in that cell is:

$$1 - \left(1 - \left(\frac{1}{C}\right)\right)^N \qquad (6)$$

Now suppose we require some minimum probability such that some robots will be in any cell at any time step. Let us call this probability P_m. Then we need to set the number of robots N high enough, such that:

$$\left(1 - \left(1 - \left(\frac{1}{C}\right)\right)^N\right) > P_m \qquad (7)$$

Now we can easily compute the number of robots N that we need to satisfy the constraints imposed by the task.

VI. HARDWARE IMPLEMENTATION

Moving from the simulation world to the real–world application is not necessarily a small step. Before we can do this move, we need to carefully choose the real–world equivalents for the two most important simulation measuring units. First, the unit to measure distance in the simulation world is 1 pixel, and we translate that to 2.54 centimeters in the real world. Second, time is measured in "steps" in the simulation, and we translate that to seconds in the real world. Having defined these basic units of measure, we can easily translate the important variables, like the maximum speed of the robot and the obstacle detection response range.

The hardware experiments are performed using our *Maxelbots*. The *Maxelbot* is a robot platform equipped with a trilateration–based localization system, developed in our laboratory [20], [14], [15]. The trilateration module includes three acoustic transducers, three parabolic cones, and a radio (RF) transceiver. The parabolic cones convert the acoustic energy into the horizontal plane. When a *Maxelbot* "pings," it emits an RF pulse from the transceiver and an acoustic pulse from one transducer. When other *Maxelbots* receive the RF pulse, they start listening for the acoustic pulse, using all three transducers. The time of flight is measured at each transducer, which is converted to distance using the speed of sound. These three distances yield the range and bearing of the "pinging" *Maxelbot*. Hence, our trilateration is a one–to–many protocol, allowing multiple *Maxelbots* to simultaneously trilaterate and determine the position of the transmitting *Maxelbot*. The effective sensing range is roughly three meters, due to attenuation of the acoustic pulse. The RF can be heard to at least ten meters.

In the following experiments the trilateration–based localization module is used in a novel way. One *Maxelbot* is moving inside the environment (the "coverage robot"), while stationary *Maxelbots* ("monitor robots"), embedded in the walls of the environment, record its position at all times.

The trilateration module on the moving *Maxelbot* emits one RF–acoustic pulse every 120 milliseconds. This translates to 8.33 pulses per second or a frequency of 8.33 Hz. This is important in order to compute the time passed from the number of trilateration readings. For example, 100 trilateration readings are equivalent to roughly 12 seconds of run time. To detect obstacles, the moving *Maxelbot* is equipped with our Obstacle Detection Module [14], which, in this setup, uses only two Infra Red sensors. The two IR sensors are placed at the front corners of the *Maxelbot* and are aimed straight ahead. We use two IR sensors instead of one (as in the simulation) to enhance wall detection and minimize collisions with the walls. The robot stops and turns if either sensor detects an object within the response range. For our experiments the IR sensors are polled once per second. With maximum battery charge, the *Maxelbot* travels approximately 22.86 centimeters per second. To prevent collisions with the walls, the response range is set to 34.29 centimeters (to cope with robot momentum). The main body of code is presented in Listing 6 in Appendix A.

The embedded stationary *monitor robots* monitor the position of the single moving *coverage robot* and send the data in real time to a PC over a serial wired connection (this was eventually replaced with a wireless Bluetooth connection, as described in [15]). This data allows us to verify that the moving robot does indeed visit all the "cells" equally often. All the environments presented next are broken into virtual cells, all cells having the same size, which is 0.762 meters × 0.762 meters. Each of the corners of these virtual cells have their coordinates preset. The number of visits to one of these virtual cells translates to how many trilateration readings were performed inside the perimeter of the cell. This allows us not only to count how many times the perimeter of any cell was crossed, but also the amount of time the robot has spent inside the cell perimeter.

Next, in this section we present three different environments with increased complexity that were chosen for these experiments. For each environment we present the experimental setup and how well uniformity is achieved by looking both at the Kullback–Leibler divergence metric and a graphical off–line rendering of the frequency of the cell occupancy.

A. Square Environment

The first environment in which we test our uniform coverage algorithm is a small, 1.524 meters × 1.524 meters, square environment. As mentioned earlier, the cell size is 0.762 meters, which yields a 2 × 2 cell environment. In this environment all cells are equivalent. The experiment serves as a control study to ensure that no unexpected biases have occurred in our hardware implementation.
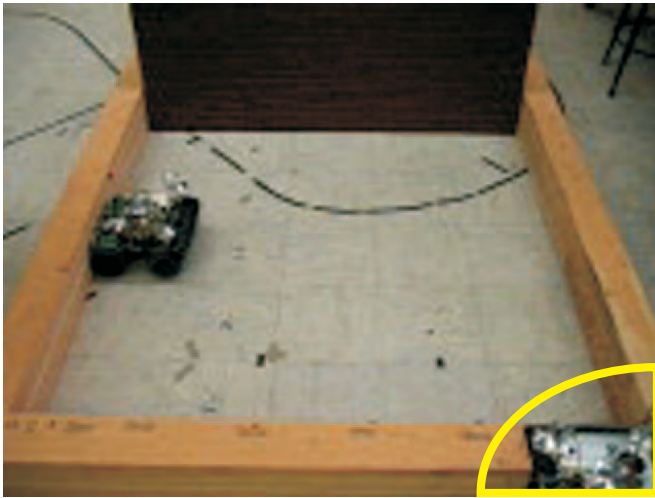
Fig. 8.   Square environment (1.524 meters × 1.524 meters)

Figure 8 shows a picture of the experimental setup that we built in our Distributed Robotics Laboratory at the University of Wyoming. Each tile on our laboratory floor is exactly 30.48 × 30.48 centimeters, and this makes it easy for the placement of the "walls", which are pine wood beams, 4.572 meters long, 15.24 centimeters wide, and 25.4 centimeters tall. For this particular experiment we also used a top of a wood table, which is exactly 1.524 meters long, for one of the walls. In the lower right corner of the picture there is one stationary monitor robot (inside the outline) that is performing the localization, based on the pulses emitted by the other moving coverage robot (shown inside the square environment). The monitor robot is connected to a serial port on a PC in order to transmit real–time data on the position it senses the coverage robot.

Based on simulation results, we estimated that 20 minutes would be sufficient to yield a K–L divergence less than 0.01. This generated a little over 9,700 trilateration readings. Table II shows the virtual cell division for this environment, which was used during the data analysis. A trace of the path taken by the moving *Maxelbot* is shown in Figure 9. The snapshots were taken at 0.2, 1, 3 and 20 minutes and show that after only 3 minutes, most of the environment was already covered.

Another useful graphical representation of this data is shown in Figure 10. Each of the four snapshots were taken at the same time the path trace snapshots in Figure 9 were taken, and show the frequency of occupancy of the four cells (brighter means the cell was visited more often). The exact number of cell visits for each cell are shown in Table III.

This shows that our cell occupancy distribution is very close to the uniform distribution and that there do not appear to be any biases in our implementation. The evolution over time of this metric is shown in Figure 11. The results indicate that the K–L divergence dropped to 0.01 after only 5,000 steps and then dropped further to 0.0012.

B. Rectangular Environment

For a more difficult experiment we used a larger 1.524 meters × 4.572 meters environment. This configuration yields

TABLE II
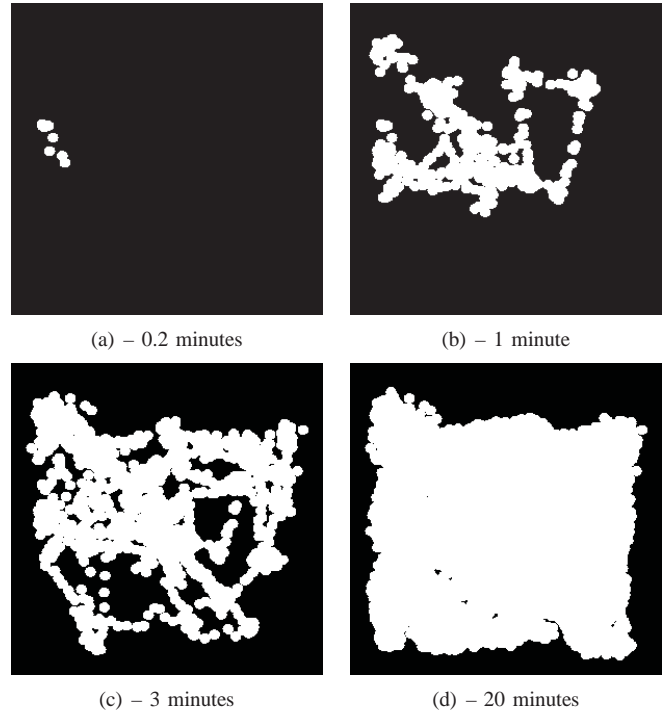CELL DIVISION OF THE 1.524 METERS × 1.524 METERS ENVIRONMENT

3	1
2	0

(a) – 0.2 minutes

(b) – 1 minute

(c) – 3 minutes

(d) – 20 minutes

Fig. 9.   *Maxelbot* path trace in the 1.524 meters × 1.524 meters environment – snapshots taken at the above shown times

(a) – 0.2 minutes
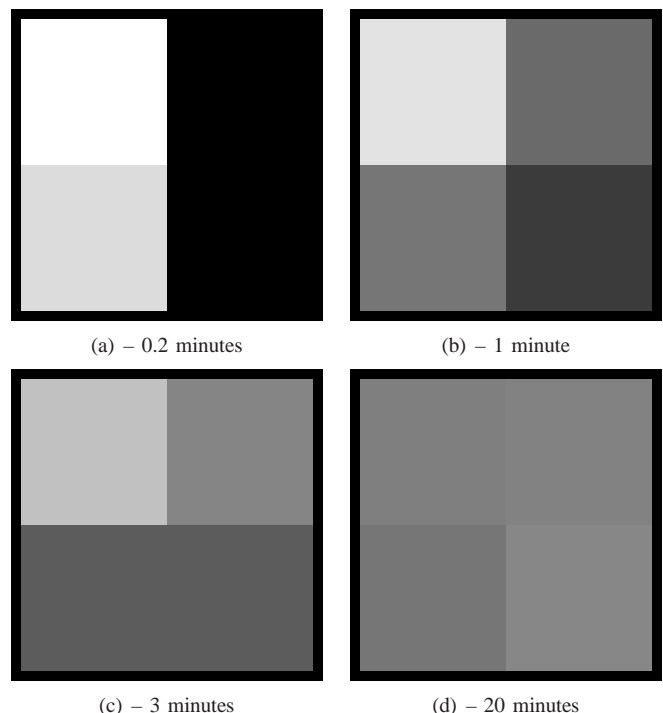
(b) – 1 minute

(c) – 3 minutes

(d) – 20 minutes

Fig. 10.   Cell occupancy in the 1.524 meters × 1.524 meters environment – snapshots taken at the times shown above

TABLE III
SPECIFIC CELL VISITS FOR THE 1.524 METERS × 1.524 METERS
ENVIRONMENT

Cell number	Time – minutes (readings × 100)			
	0.2 (1)	1 (5)	3 (15)	~20 (97)
0	0	58	270	2,576
1	0	104	391	2,464
2	43	116	271	2,253
3	57	222	568	2,407
K–L div.	–	0.110353	0.050299	0.001158
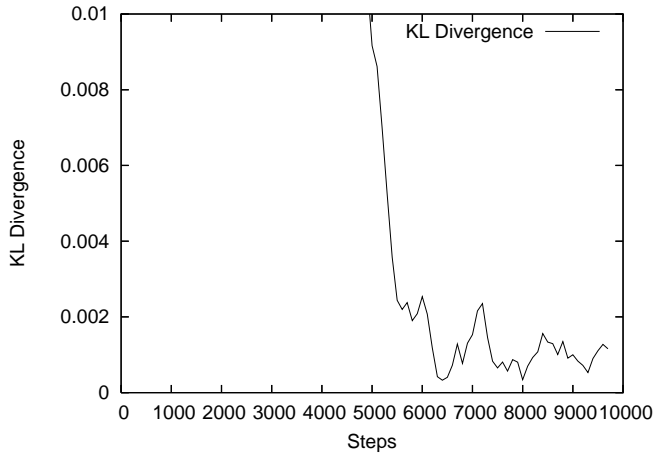
Fig. 11.   K–L divergence for the 1.524 meters × 1.524 meters environment

a 2 × 6 cell environment (with 0.762 meters cell size). Not only is the environment three times larger but it is also composed of two categories of cells – corner cells and edge cells. Table IV shows how we numbered each of the 12 cells. Although it is possible for one monitor robot to "hear" the RF–acoustic pulse generated by the moving coverage robot, the accuracy of the localization reduces drastically above three meters. The solution to this problem is to use two monitor robots, one at each end of the rectangular environment. The picture of the experimental setup is shown in Figure 12. The two *Maxelbots* used for monitoring are highlighted with the circular outlines and the coverage robot is featured in the center of the environment. Both monitor robots monitor the whole environment.

TABLE IV
CELL DIVISION OF THE 4.572 METERS × 4.572 METERS ENVIRONMENT
(THE UPPER *Maxelbot* MONITORS CELLS 0, 1, 2, 6, 7, AND 8 AND THE
LOWER *Maxelbot* MONITORS CELLS 3, 4, 5, 9, 10, AND 11)

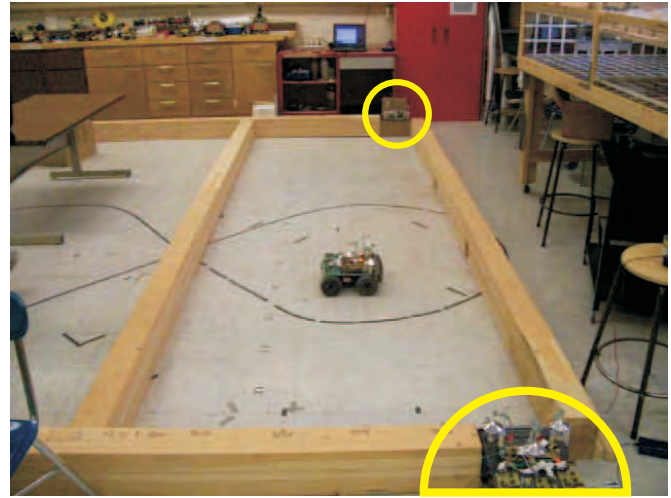|	
6	0
7	1
8	2
9	3
10	4
11	5

Fig. 12.   Rectangular environment (1.524 meters × 4.572 meters)

During this experiment there are two sets of trilateration readings that are generated (one from each monitor robot). Since data can be lost occasionally, when sent over the RF, it is possible for the two data sets to lose synchronization. This would make it very difficult to merge the data from the two monitor robots. In order to help with the synchronization between the two data sets, we introduce an identification number to the RF signal. Each time the coverage robot is broadcasting the RF – acoustic pulse combination, a number is also transmitted over the RF. This number is incremented with every pulse. The monitor robots attach this number to the trilateration reading performed. This tuple is then transmitted in real time to a PC that is connected to both monitor robots with serial cables.

Since the environment is larger, we estimated that the coverage robot would need to run for approximately two hours in order to achieve an acceptable level of uniformity (K–L divergence near 0.01). During this time, 60,000 trilateration readings were performed by each of the two monitoring *Maxelbots*. Figure 13 shows a graphical representation of the two data sets before merging. We found this to be a good test of the range and accuracy of our localization system. The localization system on one of the *Maxelbots* performs better than the other one, especially at distances over three meters (note especially the impossible locations outside of the environment).

To merge the data and reduce noise, we divided the environment into halves. Then we selected coordinates from the upper monitor robot that are in the upper half of the environment. Similarly we selected coordinates from the lower monitor robot that are in the lower half. We used the time stamp to resolve ambiguities. The results are shown in Figure 14. Although we do not filter the points shown in this Figure, for data analysis we discard all the points that are outside the outline of the environment.

The frequency of the occupancy of the cells for this environment is shown in Figure 15. The brighter the cell the higher the number of visits to that cell. The snapshots are taken at 0.2, 5, 30, 60, and 120 minutes and the exact number of visits

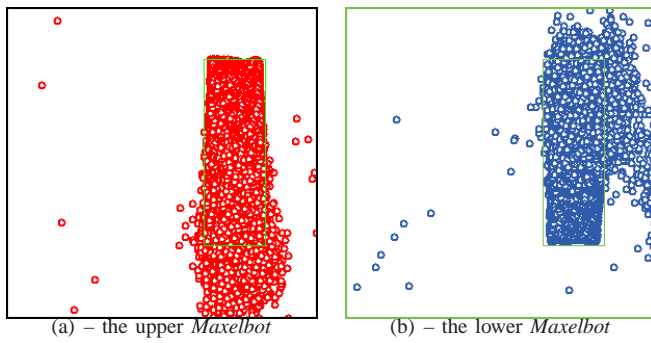(a) – the upper *Maxelbot*   (b) – the lower *Maxelbot*

Fig. 13.   Individual localization readings performed by the two fixed *Maxelbots*
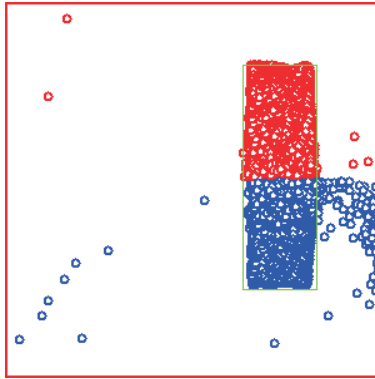
Fig. 14.   Merged localization readings performed by the two fixed *Maxelbots*

(a) – 0.2 minutes   (b) – 5 minutes   (c) – 30 minutes

(d) – 60 minutes   (e) – 120 minutes

Fig. 15.   Cell occupancy in the 4.572 meters × 1.524 meters environment – snapshots taken at 0.2, 5, 30, 60, and 120 minutes

they represent are shown in Table V. After approximately 5 minutes into the experiment, all the 12 cells in the environment were reached, with cells in the center having a larger number of visits compared with the cells close to the two ends. As the data shows, this discrepancy is drastically reduced 120 minutes into the experiment.
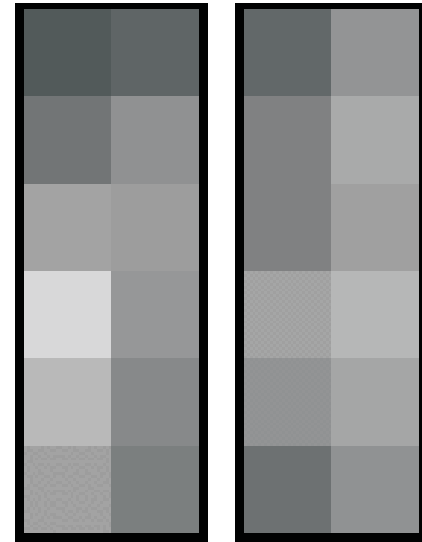
The final Kullback–Leibler divergence metric was approximately 0.015. The graph of this metric for the entire length of this experiment is shown in Figure 16. Although this wasn't quite as low as expected, it is still very reasonable. Note that there are no obvious biases towards edge or corner cells, which is good. It is clear that the initial placement of the robot is still influencing the results, so steady state has not yet been reached. Note, however, that the K–L divergence metric depends on the cell granularity. For example, if this environment is treated as being composed of three cells of size 1.524 meters × 1.524 meters (an upper, center, and lower cell), the K–L divergence drops considerably to 0.002. Decreasing the size of cells means it takes longer to achieve uniformity.

C. L–Shaped Environment

We continue testing our uniform coverage algorithm in a more complex and larger environment, specifically an L–shaped concave environment. Not only does this environment contain three categories of cells (corner, edge, and an interior cell), but achieving uniform coverage on concave environments is considered to be difficult, if not impossible [7]. The
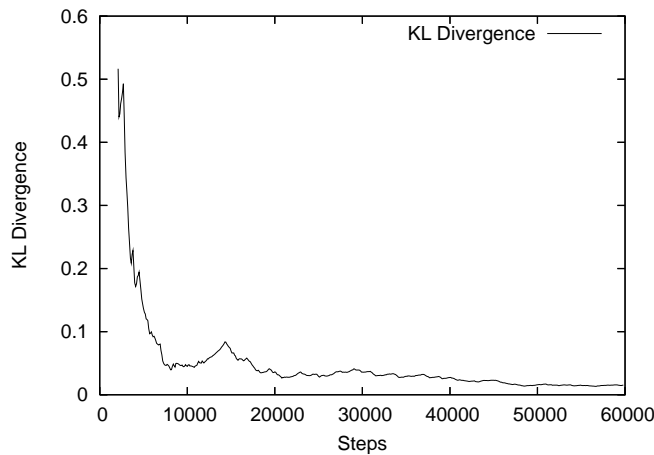
TABLE V
SPECIFIC CELL VISITS AND K–L DIVERGENCE FOR THE
1.524 METERS × 4.572 METERS ENVIRONMENT

Cell number	Time – minutes (trilateration readings × 100)				
	0.2 (1)	5 (25)	30 (150)	60 (299)	∼120 (598)
0	0	35	779	1616	5242
1	0	49	899	2472	6102
2	0	24	1131	2701	5665
3	0	256	1355	2580	6612
4	5	402	1437	2327	5908
5	0	265	1561	2103	5168
6	0	41	660	1397	3512
7	0	6	572	1944	4487
8	0	82	1105	2800	4489
9	41	638	2034	3951	5807
10	54	498	2005	3248	5206
11	0	204	1462	2761	3877
K–L div.	–	0.469953	0.069357	0.035933	0.015255

Fig. 16.   K–L divergence metric for the 4.572 meters × 1.524 meters environment

TABLE VI
CELL DIVISION OF THE L−SHAPED, 4.572 METERS × 4.572 METERS, ENVIRONMENT

19	18	17	16	15	14
13	12	11	10	9	8
6	7				
4	5				
2	3				
0	1				

Fig. 17.   Merged trilateration readings from three *Maxelbots* for the L–shaped, 4.572 meters × 4.572 meters, environment

Fig. 18.   L–shaped, 4.572 meters × 4.572 meters, environment

two sides of the L–shape are each 4.572 meters long, and the width is constant at 1.524 meters. With a cell size of 0.762 meters, this environment yields a total of 20 equal cells. The cell division and how the cells are numbered is shown in Table VI.

The picture of the experimental setup is shown in Figure 18. To monitor the location of the coverage robot in this larger environment, three monitor robots are used (highlighted with circles). As before, each monitor robot monitors a particular portion of the environment. This helps to ensure that we are receiving accurate trilateration readings. Since this environment is larger, the coverage robot was run for four hours, which was the maximum life of the robot battery. The environment outlined in Figure 17 shows the three regions, as well as a graphical representation of the approximately 118,000 merged readings, that were collected during the four hour long experiment. The positions of the three monitor are shown by the three circles located outside the perimeter and are labeled "A3", "U2", and "C5". We use these labels to distinguish between the seven *Maxelbots* we have. C5 is responsible for cells 0, 1, 2, 3, 4, and 5. *Maxelbot* U2 monitors cells 6, 7, 11, 12, 13, 17, 18, and 19. Finally, we used the data collected by *Maxelbot* A3 for cells 8, 9, 10, 14, 15, and 16.

The number of visits to each of the cells can be seen both as specific numbers in Table VII, and as a graphical representation in Figure 19. For this environment it takes the moving *Maxelbot* about 7 minutes to visit all the cells.
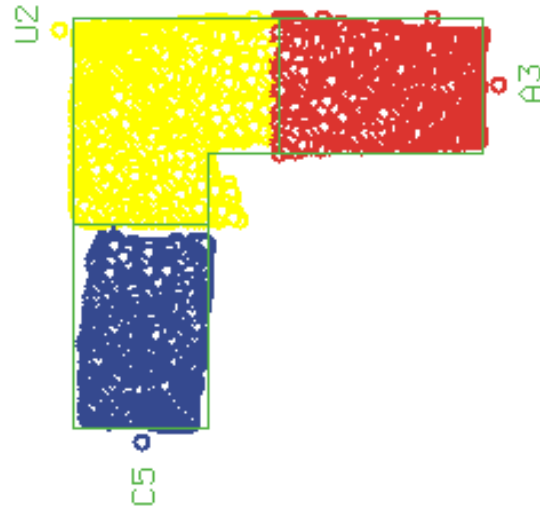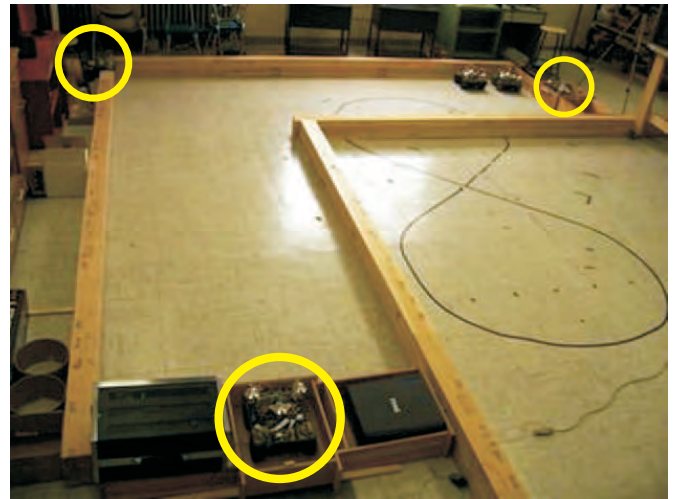
The final K–L divergence from the uniform distribution is 0.0088, which is excellent. There are no obvious biases towards any of the particular categories of cells. As shown in Figure 20, the K–L metric drops suddenly after approximately 30 minutes, meaning that a good uniform coverage is already achieved. After that, the K–L continues to drop, but at a much smaller rate. One very nice aspect of this experiment was that as the battery power dropped, the speed of the robot also dropped considerably. However, despite this real–world condition, the K–L divergence was not adversely affected.

VII. SUMMARY

In this article we have summarized different prior approaches to robotic coverage. Each of these approaches has

TABLE VII
SPECIFIC CELL VISITS AND K–L DIVERGENCE FOR THE L–SHAPED, 4.572
METERS × 4.572 METERS, ENVIRONMENT

Cell number	Time – minutes (readings × 100)			
	0.2 (1)	5 (25)	25 (125)	~240 (1180)
0	0	0	430	5547
1	0	0	701	5313
2	0	0	286	6156
3	0	0	430	6228
4	0	34	396	5721
5	0	56	342	7164
6	0	121	628	6291
7	0	90	332	5140
8	40	674	1211	7145
9	0	392	1366	6737
10	0	77	964	7314
11	0	108	619	7361
12	0	163	643	6646
13	0	124	607	5200
14	59	259	653	5898
15	0	117	862	6372
16	0	106	995	7931
17	0	28	396	5090
18	0	132	597	5978
19	0	135	655	5292
K–L divergence	–	–	0.090440	0.008751
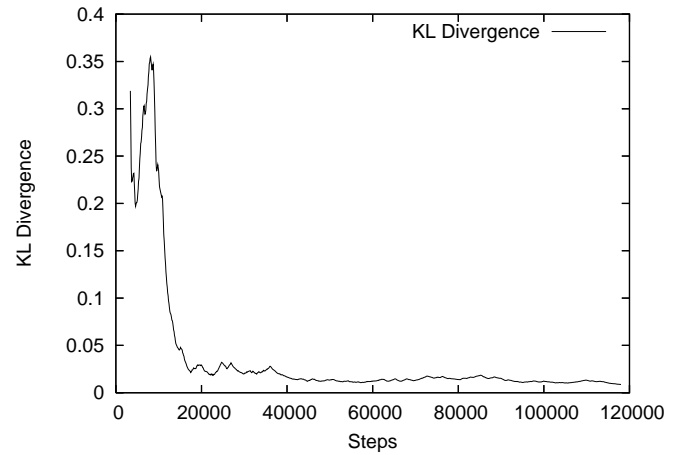
Fig. 20.   K–L divergence metric for the L–shaped, 4.572 meters × 4.572 meters, environment

(a) – 0.2 minutes

(b) – 5 minutes

(c) – 25 minutes

(d) – 240 minutes

Fig. 19.   Cell occupancy in the L–shaped 4.572 meters × 4.572 meters environment – snapshots taken at 0.2, 5, 25, and 240 minutes

its own advantages and disadvantages. Traditional complete coverage algorithms offer the guarantee that an environment is completely covered, but they require more expensive localization hardware. Some of them also require knowing the shape of the environment to be covered in advanced. Also, prior research has shown that heuristic and randomized strategies are more effective solutions when expensive sensor hardware fails to perform with 100% accuracy.

One difficulty with any approach that does not include environmental knowledge is that it is impossible to fully characterize transient behavior, such as how long it will take for a coverage robot to move from one end of an environment to another. For example, the "22cell" environment in Appendix A contains two chambers connected via a corridor. A robot will generally spend a lot of time within a chamber before crossing to the other. One nice aspect of our algorithm is that we can encourage longer traversals simply by increasing the mean free path constant f. Hence the robot will transition between one chamber and another more often. There will be a degradation in the uniformity of a coverage, but this might be worthwhile if one is trying to spot an intruder. Hence f can be modified to provide a balance between uniformity and long traversals.

Another advantage of our algorithm is that it is analyzable if the environment is known (either a priori, or via "SLAM" techniques [13]). At this point the Q matrix can be generated and transient behavior can be computed. This is generally more difficult for heuristic techniques.

In summary, our uniform coverage algorithm overcomes the limitations of prior approaches. It is a randomized theory–based algorithm that is complete, unpredictable, and covers the periphery. The algorithm requires no knowledge about the environment. Estimating the cell size can be based on the robot platform size (for cleaning operations) or sensing capabilities (for surveillance operations), rather than environmental considerations. Our algorithm also extends naturally to multiple robots. The performance of our algorithm was shown both in simulation and real–world experiments. The Kullback–Leibler divergence metric we used confirms that our algorithm is very close in achieving optimal uniform coverage.
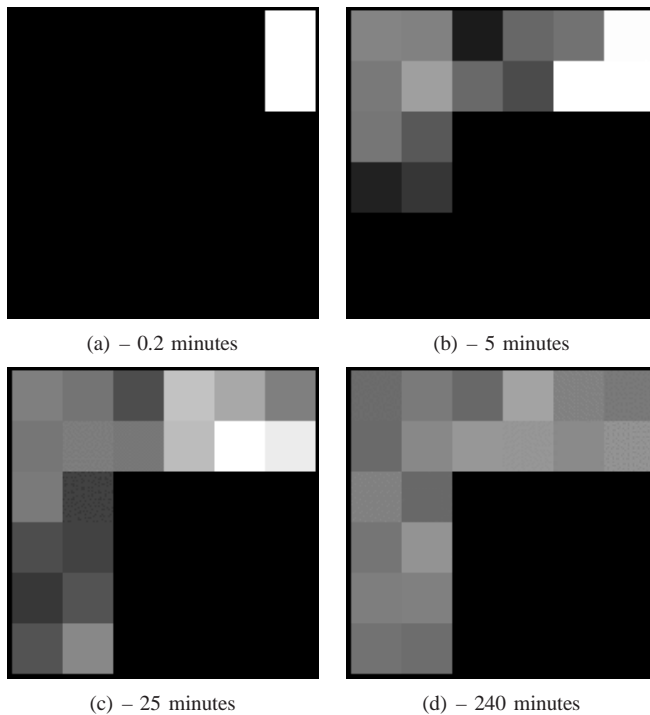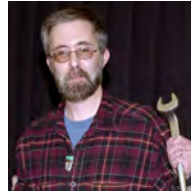
REFERENCES

[1] Anderson-Sprecher, R. Statistics Department, University of Wyoming, Personal Communication, 2008.
[2] Balch, T. (2000). The Case for Randomized Search. In *Workshop on Sensors and Motion, IEEE International Conference on Robotics and Automation, San Francisco, CA*.
[3] Choset, H. (1997). Coverage Path Planning: The Boustrophedon Cellular Decomposition. In *International Conference on Field and Service Robotics*.
[4] Choset, H. (2001). Coverage for Robotics - A Survey of Recent Results. *Annals of Mathematics and Artificial Intelligence. 31*(1-4), 113–126.
[5] Current, J. T. and D. A. Schilling (1989). The Covering Salesman Problem. In *Transportation Science*, Volume 23.
[6] Fiorini, P. and E. Prassler (2000). Cleaning and Household Robots: A Technology Survey. *Autonomous Robots 9*(3), 227–235.
[7] Gage, D. (1993). Randomized Search Strategies with Imperfect Sensors. In *Proceedings SPIE Mobile Robots VIII*, pp. 270–279.
[8] Grinstead, C. M. and J. L. Snell (1997). *Introduction to Probability* (2nd ed.)., Chapter 11, pp. 405–470. AMS.
[9] Halliday, D. and R. Resnick (1977). *Physics*, Chapter 24, pp. 522–523. John Wiley & Sons, Inc.
[10] Huang, W. H. (2001). Optimal Line-sweep-based Decompositions for Coverage Algorithms. In *International Conference on Robotics and Automation*, pp. 27–32.
[11] iRobot Corp. (2007). iRobot Roomba 500 Series Owner's Manual. Technical report, iRobot Corporation.
[12] Kullback, S. and R. A. Leibler (1951). On Information and Sufficiency. In *Annals of Mathematical Statistics*, Volume 22, pp. 79–86.
[13] Leonard, J. J. and H. F. Durrant-whyte (1991). Simultaneous Map Building and Localization for an Autonomous Mobile Robot. In *IEEE/RSJ International Workshop on Intelligent Robots and Systems* pp. 1442-1447.
[14] Maxim, P. M. (2008). *An Implementation of a Novel Localization Framework for Robots and its Application to Multi-Robot Tasks*. Ph.D. thesis, University of Wyoming, Laramie, WY.
[15] Maxim, P. M. et al. 2009. Robotic Chain Formations. In *Proceedings of the IFAC Workshop on Networked Robotics* pp. 19–24.
[16] Munkres, J. (2000). *Topology, Second Edition*, Chapter 24, pp. 155. Prentice-Hall.
[17] Mutze, U. Fundamental Theorem of Calculus in R^n, available at http://www.ma.utexas.edu/mp_arc/c/04/04-165.pdf.
[18] Partan, J. University of Massachusetts, Personal Communication, 2008.
[19] Olszta, P. W. (2008). The Free OpenGL Utility Toolkit. Technical report, X.Org Foundation. http://freeglut.sourceforge.net/, Accessed on September 23, 2008.
[20] Spears, W. M. et al. (2006). Where Are You? In Şahin, E., Spears, W., eds.: *Swarm Robotics, Springer-Verlag*.
[21] Stark H. and J. Woods (1986). *Probability, Random Processes, and Estimator Theory for Engineers*, Chapter 8, pp. 316–317. Prentice-Hall.

William M. Spears received a Ph.D. in Computer Science from George Mason University in 1998. He has an international reputation for his expertise in evolutionary computing and has a published book on the topic. He has co-edited books on swarm robotics and evolutionary computation. His current research includes distributed robotics, the epidemiology of virus spread, evolutionary algorithms, complex adaptive systems, and learning and adaptation. He was co-founder of the University of Wyoming Distributed Robotics Laboratory and has approximately 75 publications. He is the CEO of Swarmotics, LLC. He can be contacted at wspears@swarmotics.com.

APPENDIX A

The main while loop of the *Maxelbot* is shown in Listing 6. Every second the obstacle avoidance module is polled and a move counter is incremented. If an object is detected within the sensor response range, a random turn is made clockwise or counterclockwise, and the move counter is reset. If the move counter is equal to two, a random turn is also made (corresponding to the mean free path length). Otherwise the robot continues to move forward.

Listing 6. The main body of the *Maxelbot* source code for the uniform coverage algorithm

```
   while (1)
   {
      // 1 second delay
      delay( 150 );
5     move_counter++;
      // get a fresh reading from the OAM
      ask_the_OAM( OAM_I2C_ADDRESS );
      // wait for ALL the data to be received
10    // from the OAM
      while( i2c_com_in_progress );
      while( get_inches( sensors[0] ) <= SENSOR_RANGE ||
             get_inches( sensors[1] ) <= SENSOR_RANGE ||
             move_counter == 2  ) // Mean free path
15    {
         // rand() returns values from 0 to 32767
         degs = ( int )( rand() % 180 + 1 );
         if ( degs % 2 ) // Direction of turn
           dir = 1;
20       else
           dir = 0;
         turn ( dir, degs );
         move_counter = 0;
         ask_the_OAM( OAM_I2C_ADDRESS );
25       while( i2c_com_in_progress );
      }
      // Move forward
      forward( 1, 20 ); // Left motors
      forward( 0, 20 ); // Right motors
30 }
```

The output of the environment generator presented in Figure 2 is shown in Listing 7. First, the total number of vertices is shown, followed by the coordinates of each vertex, as selected by the user. This environment is split into 47 equal-size square cells, each cell being 20×20 units (cell size is shown on the last line of the output). The coordinates of the lower left corner for each of the 47 cells of this environment are then presented. No decrement factor was used in this example.

Paul M. Maxim received a Ph.D. in Computer Science from University of Wyoming in 2008. He has co-authored several publications and two book chapters during his doctoral research. As a member of the University of Wyoming Distributed Robotics Laboratory, he was the main developer of the Maxelbot robot platform. His research interests include distributed robotics and robot design and development. He is currently employed by Wyoming Department of Transportation – Intelligent Transportation Systems program where he is mainly involved with hardware integration, as well as software designed and development. He can be contacted at paul.maxim@dot.state.wy.us.

Listing 7. Data generated by the environment generator for the arbitrary environment presented in Figure 2

```
     vertex_number          12
     40      120
     40      260
     120     260
5    120     160
     200     160
     200     200
     260     200
     260     100
10   200     100
     200     140
     120     140
     120     120
     cell_number            47
15   40      120
     40      140
     40      160
     40      180
     40      200
20   40      220
     40      240
     60      120
     60      140
     60      160
25   60      180
     60      200
     60      220
     60      240
     80      120
30   80      140
     80      160
     80      180
     80      200
     80      220
35   80      240
     100     120
     100     140
     100     160
     100     180
40   100     200
     100     220
     100     240
     120     140
     140     140
45   160     140
     180     140
     200     100
     200     120
     200     140
50   200     160
     200     180
     220     100
     220     120
     220     140
55   220     160
     220     180
     240     100
     240     120
     240     140
60   240     160
     240     180
     cell_size              20
```
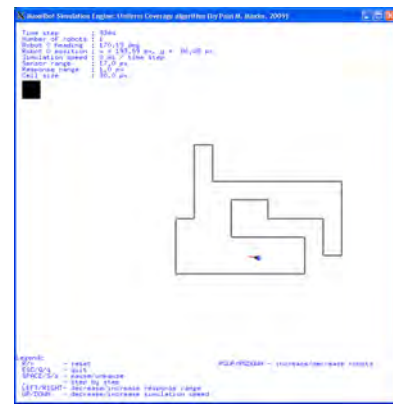
Our uniform coverage algorithm was tested in simulation on numerous environments. In Section IV we present the experimental results. The outlines of the tested environments are presented in Figures 21, 22, and 23. Another environment example is shown in Figure 3.

(a) – 3x1 (cell size of 30)

(b) – 4cellL (cell size of 30)

(c) – 4cellRoll (cell size of 30)

Fig. 21.   Examples of environments tested with the uniform coverage algorithm

(a) – 7cell (cell size of 30)

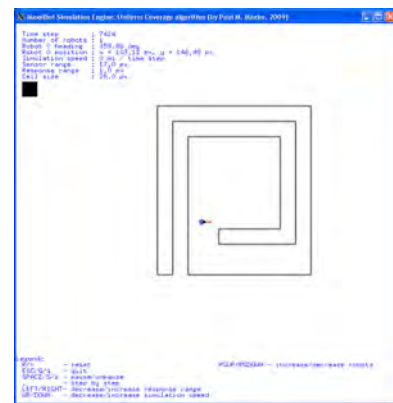(a) – 35cell (cell size of 30)

(b) – 3x3 (cell size of 30)

(b) – 36cell (cell size of 30)

(c) – 15cell (cell size of 30)

(c) – 82cell (cell size of 25)

(d) – 22cell (cell size of 30)

(d) – 10x10 (cell size of 30)

Fig. 22.   Examples of environments tested with the uniform coverage algorithm

Fig. 23.   Examples of environments tested with the uniform coverage algorithm